ELSEVIER

Computational Geometry 11 (1998) 125–141

Computational Geometry
Theory and Applications

Offset-polygon annulus placement problems ☆

Gill Barequet [a,*], Amy J. Briggs [b], Matthew T. Dickerson [b], Michael T. Goodrich [a]

[a] *Center for Geometric Computing, Department of Computer Science, Johns Hopkins University, Baltimore, MD 21218, USA*
[b] *Department of Mathematics and Computer Science, Middlebury College, Middlebury, VT 05753, USA*

Communicated by J.-R. Sack; submitted 26 June 1997; accepted 5 July 1998

Abstract

An offset-polygon annulus region is defined in terms of a polygon P and a distance $\delta > 0$ (offset of P). In this paper we solve several containment problems for polygon annulus regions with respect to an input point set. Optimization criteria include both maximizing the number of points contained in a fixed size annulus and minimizing the size of the annulus needed to contain all points. We address the following variants of the problem: placement of an annulus of a convex polygon as well as of a simple polygon; placement by translation only, or by translation and rotation; off-line and on-line versions of the corresponding decision problems; and decision as well as optimization versions of the problems. We present efficient algorithms in each case. © 1998 Elsevier Science B.V. All rights reserved.

Keywords: Optimal polygon placement; Tolerancing; Robot localization; Offsetting

1. Introduction

We begin with intuitive descriptions and motivations for the studied problems and then give some more formal definitions.

1.1. Background and applications

In this paper we address several variants of the problem of placing an annulus defined by offsetting a given polygon such that it covers all (or a maximum number of) points of a given set of points. This problem is motivated by several applications. For example, in the *robot localization* problem (see, e.g., [14]), a robot should determine its current location in some environment map from a set of points

☆ Work on this paper by the first and the fourth authors has been supported in part by the U.S. Army Research Office under Grant DAAH04-96-1-0013. Work by the third author has been supported in part by the National Science Foundation under Grant CCR-93-1714. Work by the fourth author has been supported also by NSF grant CCR-96-25289. A preliminary version of this paper appeared in WADS '97.

* Corresponding author. E-mail: barequet@cs.jhu.edu.

0925-7721/98/$ – see front matter
PII: S0925-7721(98)00025-X

obtained by a distance range sensor. Due to the inherent errors in range finding (noisy data as well as errors in measurements), the points usually do not define an exact match. Most points, however, fall within some distance $\delta > 0$ of the environment boundary. Thus the localization problem can be viewed as finding some *optimal* placement of the environment model (typically a polygon) with respect to the set of points and a distance $\delta > 0$. The goal is to maximize the number of points corresponding to a corridor or annulus region of δ around the walls. This problem is well-modeled as an instance of the offset-polygon maximum-cover problem, defined in Section 1.3. Our method is not susceptible to noisy data, whereas other approaches, such as least squares [8] or minimizing the annulus to contain all points, are more sensitive to such noise. Moreover, our method generalizes also to rotations of the annulus. A second application is a pattern matching problem arising in computer vision (see, e.g., [16]), where the input consists of a set of points taken from some image and a pattern (polygon) that one would like to locate in this image. A good match can be found by determining a placement of the polygon that maximizes the number of points within some distance $\delta > 0$ of the image points. Yet another application arises in geometric tolerancing. Chang and Yap [6] describe geometric tolerancing as being concerned with the specification of geometric shapes for use in manufacturing of mechanical parts, and they note that, since manufacturing processes are inherently imprecise, it is imperative that such geometric designs be accompanied by tolerance specifications. An instance of the tolerancing problem is to take a set of points representing an actual measurement of a manufactured object (using a coordinate-measuring machine, laser range-finder, or scanning electron microscope [11]) and determine whether the manufactured object matches a polygon (the design) within some tolerance $\delta > 0$. This corresponds, for example, to the *tolerance zone* semantics described by Requicha [20], Srinivasan [22] and Yap [24].

1.2. Previous related work

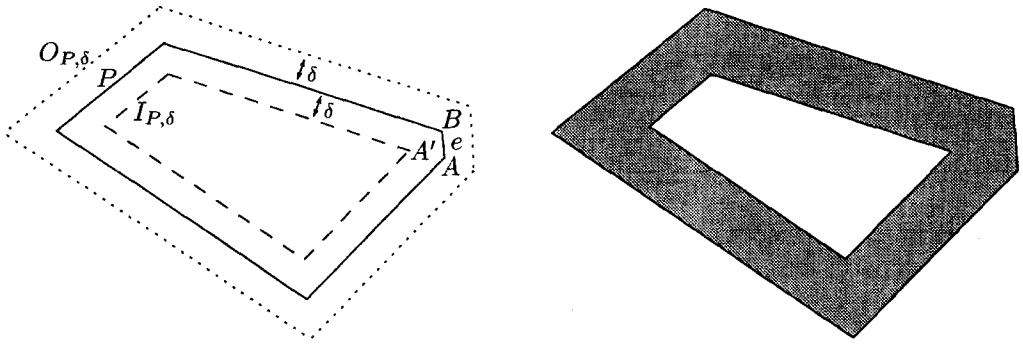
The notion of polygon annulus placement relative to a set of points appears to be new in the computational-geometry literature. There are nevertheless several related problems that have been studied before, including variants directed at placing an entire polygon (not an annulus) to cover a set or subset of points (see, e.g., [5,10,12,13]). These problems are quite interesting, but they do not model important aspects of optimizing polygon placement (as mentioned in the applications above). Previous work directed at annulus problems, on the other hand, have dealt exclusively with circular annuli (see, e.g., [2,3,9,15,18,21,23]). These characterizations capture well the notion of "roundness" present in a set of points, but they do not easily extend to polygonal shape matching.

1.3. Definitions and problems

We start with definitions for convex polygons to simplify the presentation. Extensions to simple polygons are made in Section 5.

Definition 1 (Offset annulus). The *δ-annulus* of a convex polygon P is the closed region defined by all points in the plane at distance at most δ from the boundary of P.

Definition 2 (Offset polygons). Given a convex polygon P and a distance $\delta > 0$, the *δ-offset* polygons are defined as follows. The *inner* δ-offset polygon $I_{P,\delta}$ is the boundary portions of the δ-annulus of P that are properly contained by P. Similarly, the *outer* δ-offset polygon $O_{P,\delta}$ is the boundary portions of the δ-annulus of P outside of (i.e., properly containing) P.



(a) (b)

Fig. 1. Offsetting a polygon. (a) Linearized inner and outer δ-offset polygons. (b) The δ-annulus region.

Note that $I_{P,\delta}$ is made up of edges that are parallel to edges of P (although there may be some edges of P that are not parallel to any in $I_{P,\delta}$). The offset polygon $O_{P,\delta}$, on the other hand, is made up of alternating line segments and circular arcs, and every edge of P is parallel to some edge of $O_{P,\delta}$. One can also imagine a fully *linearized* version of the outer offset polygon, where one extends each of the linear edges until they meet the extensions of neighboring linear edges. For simplicity of presentation, we will first discuss algorithms for solving polygon-annulus problems adopting this linearized view, and we will then show how to extend these to the more-natural standard notion of a δ-offset without affecting the running times by more than a constant factor.

Fig. 1(a) shows a convex polygon P (with solid edges) and its inner and linearized outer offset polygons $I_{P,\delta}$ and $O_{P,\delta}$ (with dashed and dotted edges, respectively) for some value of δ. Note that for any convex polygon P and for any value of δ the outer offset polygon $O_{P,\delta}$ always has the same number of edges as P, but the inner offset polygon $I_{P,\delta}$ may have fewer edges. In this example the edge $e \in P$ does not have a counterpart in $I_{P,\delta}$. More specifically, the point A, edge e, and point B, all in P, collapse into a single point A' in $I_{P,\delta}$. Also, the offset polygons $I_{P,\delta}$ and $O_{P,\delta}$ are usually *not* scaled versions of P (unless P is a regular polygon). The δ-annulus region of P is shown shaded in Fig. 1(b). Note that the annulus region is defined to include the boundary edges. Although these definitions are stated for convex polygons, we show that in many cases they can easily be extended to simple polygons (see Section 5). In any case, the definition of δ-annulus regions naturally gives rise to the following problems.

- **Offset-polygon max-cover**. Given a set S of n points in the plane, a convex polygon P, and a distance δ, find a placement τ of P that maximizes the number of points of S contained in the δ-annulus region of $\tau(P)$. Report the placement τ and the set of contained points.
- **Offset-polygon containment (decision version)**. Given a set S of n points in the plane, a convex polygon P, and a distance δ, determine if there exists a placement τ of P such that *all* n points of S are contained in the δ-annulus region of $\tau(P)$. Report such a placement τ if one exists.
- **Offset-polygon containment (optimization version)**. Given a set S of n points in the plane, and a convex polygon P, find the smallest value of $\delta > 0$ such that there exists a placement τ of P with *all* n points of S being contained in the δ-annulus region of $\tau(P)$. Report such a placement τ if one exists, together with this optimal value of $\delta > 0$.

Note that we can use an algorithm for either the offset-polygon maximum-cover problem or for the width-optimization problem to solve the offset-polygon containment decision problem. In particular, the answer

for the decision problem is "yes" if and only if for the former problem the value of k—the maximum number of points contained in the δ-annulus for P—is n, or for the latter problem the value of δ'—the minimum width of an annulus that contains all the points—is at most δ.

1.4. Outline and summary of results

Let n be the number of input points and let m be the number of edges (and vertices) of the given polygon P. In this paper we give several results for solving the offset-polygon annulus maximum-cover and containment (decision and optimization) problems. We show that if we restrict the containment decision problem to convex polygons under translation only, then we can determine a containing placement of a minimum-width annulus of P, if one exists, in $O(n \log m \log(nm) + m)$ time. Our method involves a nontrivial extension of the roundness method of Duncan et al. [9] to offset polygons by using the polygon-offset nearest-neighbor and furthest-neighbor diagrams [4] and the simplest (and most practical) version of parametric searching.

We also study the offset-polygon maximum-cover problem for convex polygons under translation, showing that this more-general problem can be solved in $O(n^2 \log(nm) + m)$ time and $O(n + m)$ space. Our algorithm is a nontrivial generalization of the technique of Barequet et al. [5]. In addition, we show how to solve this problem under translation and rotation by combining this approach with an extension of the rotation-diagram technique of Dickerson and Scharstein [10]. The resulting time bound in this case is $O(n^3 \log(nm) + m)$ using $O(n + m)$ space in the worst case. Under some reasonable "fatness" conditions (which we make precise in Section 5), we show that our techniques can be generalized for simple polygons under translation to result in an algorithm running in $O(n^2 m^2 \log(nm))$ time and $O(nm^2)$ space.

In addition to the off-line results discussed above, we also describe a method, based upon an interesting dynamic data structure, that solves an on-line version of the offset-polygon containment decision problem under translation. The algorithm reads points one at a time, halting and answering "no" when a placement containing all points read so far is no longer possible, or, alternatively, running to completion on n points and answering "yes". In the worst case this on-line algorithm runs in $O(n^2 m^2 \log(nm))$ time and $O(n^2 m^2)$ space for simple polygons. For many distributions of points, however, it performs significantly better. In particular, for convex polygons our on-line algorithm runs in $O(nh \log(nm) + m)$ time and requires only $O(nh + m)$ space, where h depends on the distribution (see Section 4.3). (In the worst case $h = \Theta(n)$, but for many distributions h is substantially smaller.)

The outline of the paper is as follows. We begin in Section 2 with some important geometric properties and primitives. In Section 3 we present the algorithms for convex polygons, and in Section 4 we give our on-line solution to the offset-polygon maximum-cover problem. In Section 5 we extend our solutions to the offset-polygon maximum-cover problem to simple polygons. We conclude with Section 6.

2. Key geometric properties

An important step of our algorithms is the computation of the intersections between translated copies of offset polygons. For simplicity of expression, let us assume we are dealing with linearized offset polygons; we show later how to remove this restriction to deal with the more-standard definition of δ-annulus region with only a constant-factor increase in the running times of our algorithms. Let us

therefore consider an upper bound on the number of intersections between translated copies of linearized offset polygons, and a description of how to compute them. It is well known that two translated homothetic copies of the same convex polygon can intersect at most twice (where in the degenerate case an intersection may be a segment rather than a point). The following theorem states that translations of inner and outer offsets of a convex polygon can also intersect at most twice.

Theorem 1. *Given a polygon P, a distance δ, and a translation τ, the (linearized) offset polygons $\tau(I_{P,\delta})$ and $O_{P,\delta}$ intersect at most twice, where each intersection may be a point or (in the degenerate case) a segment.*

Proof. Given in Appendix A. □

The technique used in the proof of Theorem 1 also provides the necessary framework for the proof of Lemma 2. The weak monotonicity of the width function between chains of the two polygons (in this proof) suffices for using the *tentative prune-and-search* technique of Kirkpatrick and Snoeyink [17] to compute the intersection points.

Lemma 2. *The intersections between offset polygons $\tau(I_{P,\delta})$ and $O_{P,\delta}$ can be found in $O(\log m)$ time, where m is the number of vertices of P.*

We compute these intersections because they correspond to placements of the annulus region such that two (or more) points of S are in contact with the boundary of the annulus region.

Lemmas 3 and 4 are generalizations of lemmas from [5,10] that deal with intersections between two copies of the same polygon.

Lemma 3. *Let P be a convex polygon, q_1, q_2 points, and τ_1 and τ_2 the translations mapping the origin to points q_1 and q_2, respectively. For any point x, let $\tau_x = q_2 - x$ be the translation that maps x to q_2. Then both q_1 and q_2 are contained in the δ-annulus region of $\tau_x(\tau_1(P))$ if and only if x is contained in the intersection of the δ-annulus regions of $\tau_1(P)$ and $\tau_2(P)$.*

Lemma 3 is illustrated in Fig. 2. Translated copies of the offset annulus region are placed on the points q_1 and q_2, and a point x is shown in the intersection of the two annulus regions. The translation τ_x that maps x to q_2 also maps $\tau_1(P)$ such that $\tau_x(\tau_1(P))$ contains both q_1 and q_2. The proof of this lemma is based on simple vector arithmetic (see the references cited above). The lemma provides a method for finding translations of P that contain multiple points of S. The containing translations correspond to the intersections between copies of the annulus regions placed on the contained points. Lemma 2 tells us that the boundaries of these intersections can be computed quickly.

The following (rather simple) lemma guarantees that we can limit our search for an optimal placement to translations that have at least one point of S on the boundary (either inner or outer) of the annulus region.

Lemma 4. *Let P be a convex polygon and S be a nonempty set of points contained in the δ-annulus of P. Then there exists a translation τ such that S is contained in the δ-annulus of $\tau(P)$ and at least one point of S is on the boundary of the annulus region.*

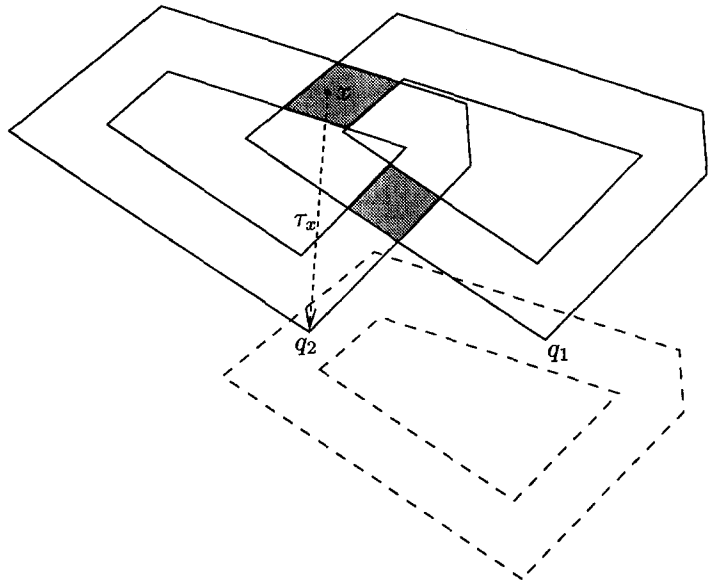


Fig. 2. Translation of an offset annulus region containing two points.

To prove this lemma, let τ be any placement containing at least one point. Assuming that there are no points on the boundary of $\tau(P)$, let $x \in S$ be the point closest to the (inner or outer) boundary. Translate $\tau(p)$ the minimal distance to put x on the boundary. This new translation contains all the points contained in $\tau(P)$ but with x on the boundary, satisfying the conditions of the lemma. We follow the terminology of [7] and denote such a placement τ as *stable* (to be used in the algorithm given in the next section).

3. Algorithms for convex polygons

We can now present algorithms for solving offset-polygon placement problems, starting with what is conceptually the simplest problem for a convex polygon shape.

3.1. Offset-polygon containment optimization under translation

We first briefly describe a deterministic $O(n \log m \log(nm) + m)$-time algorithm for solving the annulus-width optimization problem. Given a set S of n points and a convex polygon P with m vertices, find the minimum-width annulus of P that covers S. For this purpose we define the *convex polygon-offset* distance-function $\mathcal{D}_P$ that corresponds to P and compute the nearest- and furthest-site Voronoi diagrams of S with respect to $\mathcal{D}_P$ (see [4]). This can be performed in $O(n(\log n + \log^2 m) + m)$ time. [1] Next we use the method of [9] (where the authors minimize the width of a *circular* annulus) and consider the overlay of the two diagrams. As is well known, the center of the minimum-width annulus that contains S is either a vertex of one of the two diagrams (possibly a vertex at infinity in the furthest-site diagram) or

[1] A bound of $O(n(\log n + \log m) + m)$ was erroneously claimed in [4]. The corrected analysis is found in the full version of that paper.

a point of intersection between the two diagrams. Given a specific value of δ, we place reflected δ-annuli centered at all the points of S and observe (as in [9]) the overlay for determining whether the intersection of all annuli is nonempty. (The intersection contains the loci of all feasible placements of the annulus so that it covers S.) This step takes $O(n \log m)$ time. Finally, a parametric-searching algorithm is applied for optimizing (minimizing) the value of δ for which the intersection of all the annuli is nonempty. Over all, the whole procedure requires $O(n \log m \log(nm) + m)$ time.

3.2. *Offset-polygon max-cover under translation*

In this section we consider offset-polygon maximum-cover under translation. Our algorithm extends the techniques of Barequet et al. [5] to allow for containment within the annulus region rather than containment by the entire polygon. The idea is to do an anchored sweep of *both* the inner and outer offset polygons around each point of S. The critical events of the sweep occur when some point of S either enters or exits the δ-annulus. The full algorithm is given in Fig. 3.

The correctness of this algorithm follows from Lemmas 3 and 4. There exists at least one optimal placement with a point in contact with the annulus boundary, and this placement will be found by the sweep. The only additional detail deals with the processing of degenerate intersections, where the intersection between two offset polygons is a segment (along a connected portion of an edge) rather than a discrete point. In this case only one of the two endpoints of the segment corresponds to an event. If the point q_i is currently marked "in" then it is at the second endpoint of the intersection segment where it changes to "not in". Conversely for points marked "not in", it is at the first endpoint of the segment where it changes to "in". This follows from the fact that the entire segment corresponds to a translation in which both points q_i and q_j are on the boundary of the translated polygon and so points that are "in" remain so until the *end* of the segment, whereas points that are "not in" become "in" at the *start* of the segment.

We measure the complexity of our maximum-cover algorithm under translation as a function of two variables: m, the number of vertices of P, and n, the number of points in the set S. The preprocessing step requires $O(m)$ time and space for computing and storing the offset polygons $I = I_{P,\delta}$ and $O = O_{P,\delta}$. The offset polygons are stored such that later intersection tests can be performed in $O(\log m)$ time and space (see [17] and Section 2). The steps inside the inner nested loop execute $O(n^2)$ times. Since each pair of points has two offset polygons, each of which has at most two intersections with the polygon being swept, the total size of the queue is $O(n)$ and queue operations can be performed in $O(\log n)$ time. Polygon intersections in Step 6 can be computed in $O(\log m)$ time (by Lemma 3). The total running time is therefore $O(n^2 \log(nm) + m)$ in the worst case. The algorithm requires $O(n + m)$ space.

The running time of the algorithm can be improved by the use of bucketing. Suppose that we bucket all the points, where the bucket rectangles correspond (in size and orientation) to the smallest rectangle enclosing O. We now introduce a third variable k which is the maximum number of points that can be contained in a translation of O. Then the number of points in any bucket is $O(k)$ [5] and the inner loop beginning at Step 5 needs to be iterated only for those points q_j in the same bucket as q_i or in one of the 8 neighboring buckets. We note that with a standard bucketing approach the number of buckets required is $O(A_S/A_P)$, where A_S is the area of the entire region bounding the input point set S, and A_P is the area of the bucket which is proportional to the area of the polygon P. However, the only buckets that need to be explicitly initialized are those that contain points of S and their immediate neighbors, thus the preprocessing step for bucketing still requires only $O(n)$ time. The standard bucketing algorithm therefore requires $O(nk \log(mk) + m)$ time and $O(A_S/A_P + n + m)$ space in the worst case.

I. Preprocessing:

1. Preprocess offset polygons $I = I_{P,\delta}$ and $O = O_{P,\delta}$ for intersection computation.
2. Initialize a priority queue Q which will store points in clockwise order around the boundaries of the offset polygons I and O.

II. Iteration:

1. Set max := 0. {# of points in optimal placement so far}
2. **FOR** each point $q_i \in S$ **DO BEGIN** {Anchored sweep of I and O around q_i}
3. Let P' be I. {Start with anchored sweep of I}
4. Set $c := 1$. {Points contained by current translation}
5. **FOR** each $j \neq i$ and $q_j \in S$ **DO BEGIN** {Examine nearby points for containment}
6. Set $X := \{x \mid x \in \partial\tau_i(I) \cap \partial\tau_j(P')\} \cup \{x \mid x \in \partial\tau_i(O) \cap \partial\tau_j(P')\}$.
7. **FOR** all $x \in X$ **DO**
8. Add (x, j) to Q. {Add all intersections to event queue}
   END FOR
9. **IF** q_j is contained in the δ-annulus of $\tau_i(P)$ **THEN**
10. Mark q_j "in"; Set $c := c + 1$; {Mark and count points currently contained}
    ELSE
11. Mark q_j "not in".
    END IF
    END FOR
12. **WHILE** $Q \neq \emptyset$ **DO BEGIN** {Sweep with intersections as events}
13. Delete (x, j) from front of Q. {Update structures and counters}
14. **IF** q_j is "not in" **THEN** {See comments on (degenerate) intersections}
15. Set $c := c + 1$; Mark q_j "in".
    ELSE
16. Set $c := c - 1$; Mark q_j "not in".
    END IF
17. **IF** $c >$ max **THEN**
18. Set max := c; Store translation.
    END IF
    END WHILE
19. **REPEAT** Steps 4–18 with $P' = O$. {Now do an anchored sweep of O around q_i}

END FOR

Fig. 3. Max-cover algorithm under translation for convex polygons.

In case A_S/A_P is too large we can use either a degraded grid or a hashing table. By using the degraded grid approach of Lenhof and Smid [19] the bucketing can require only $O(n)$ space at the cost of an $O(n \log n)$-time preprocessing-step for sorting the input points. Alternatively, the buckets can be stored in a hashing table of size $O(n)$ and be accessed in expected $O(1)$ time (per operation). Thus by using these approaches we can solve the problem either in $O(nk \log(mk) + m + n \log n)$ time and $O(n + m)$ space in the worst case (by a degraded grid) or $O(nk \log(mk) + m)$ time and $O(n + m)$ space in the average case (with a hashing table). (Unfortunately, the value of k does not necessarily correspond to the number of points contained in a translation of the δ-*annulus* region of P, but rather to the possibly larger number of points contained within the entire outer polygon O.)

3.3. Offset-polygon max-cover under translation and rotation

We now describe how the offset-polygon maximum-cover problem can be solved for convex polygons when we allow for translations and rotations. To solve this problem we extend the results of Dickerson and Scharstein [10] and make use of their *rotation diagram* technique. We refer the reader to [10] for details on this method; here we describe only the necessary modifications in the approach and in the complexity analysis.

This method creates a rotation diagram R_{q_i} for each point q_i. The diagram R_{q_i} is a description of the configuration space of all placements of the polygon P that keep the boundary of P in contact with q_i. The horizontal axis of this diagram represents the angle of rotation (from 0 to 2π). The vertical axis represents the arclength along ∂P (from 0 to the circumference of P). For each other point q_j, the diagram R_{q_i} includes the region of all such placements that contain q_j. It is shown in [10] that this containing region for q_j is decomposable into $O(m^2)$ subregions of constant complexity. The left and right boundaries of these subregions are certain critical angles of rotation, where vertices of one polygon pass through edges of another. The upper and lower boundaries are shown to be sine curves. To solve the optimal placement problem, the algorithm performs a plane sweep of each rotation diagram R_{q_i} to find the region of greatest depth. This gives the optimal placement of P that is in contact with q_i.

The main difference for the annulus placement problem is that we need two rotation diagrams for each point q_i: one for the inner offset polygon $I_{P,\delta}$ and one for the outer offset polygon $O_{P,\delta}$. Furthermore, each of these two rotation diagrams for q_i has regions for each $q_j \neq q_i$ that represent containment in the annulus region rather than in the entire polygon. The following lemma states that these modified rotation diagrams have the same complexities.

Lemma 5. *For convex polygons, the polygon annulus containing regions for a given point is decomposable into $O(m^2)$ subregions each of which has constant complexity: vertical left and right boundaries and a sine curve for the top and bottom boundaries.*

The proofs of [10] suffice to show that the upper and lower boundaries are still sine curves. The $O(m^2)$ is a trivial upper bound on the number of subregions, which is actually attainable. There is however a constant factor increase in the complexity of the diagrams. The number of critical angles are doubled because we now count intersections of both the inner and outer offset polygons placed at point q_i and either the inner or outer polygon at q_j (depending on which rotation diagram we are computing). Therefore, since the number of subregions can double, the number of intersection points can increase by a factor of four. To solve the offset-polygon maximum-cover problem we use the same idea of the rotation diagram and perform plane sweeps of each of the $2n$ diagrams. Lemma 4 tells us that this suffices because even with a restriction to translation only there is at least one optimal placement that has a point on an inner or an outer boundary of the annulus region. The space complexity of the algorithm remains the same as in [10]: the plane sweep considers at one time only one subregion out of the $O(m^2)$ subregions that correspond to some point q_j, while the annulus is in contact with point q_i. Thus we can state the following theorem.

Theorem 6. *The convex offset-polygon maximum-cover problem can be solved for translation and rotation in $O(n^3 \log(nm) + m)$ time and $O(n + m)$ space in the worst case.*

3.4. True δ-tolerancing

As mentioned earlier, our algorithms assume a linearized outer polygon boundary. For adapting the linearized versions of the offset-polygon maximum-cover and offset-polygon containment problems to their (standard) nonlinear forms, we need only show that the framework for the offset-annulus variant works also for the true δ-tolerancing case. The key to this adaptation lies in the fact that for every convex polygon P, tolerance δ, and a translation τ, the number of intersections of $\tau(I_{P,\delta})$ and the true outer boundary $O_{P,\delta}$ is still at most two. The proof of this claim is almost identical to that of Theorem 1 (see Section 2). Indeed, the weak monotonicity of the curves is preserved (we do not need the curves to be piecewise-linear). Furthermore, we can still apply the prune-and-search technique, since the simplicity of the pieces of the curves is also maintained: it takes a constant amount of time to evaluate the intersection of a circular arc with a line segment or with another circular arc. Therefore we are able to apply the same algorithm (for the translation-only variant) as in Section 3.2 and obtain the same asymptotic running time and space. Similar arguments hold for the translation and rotation version of the problem. Careful analysis reveals that the complexity of the algorithm presented in Section 3.3 remains asymptotically the same (within a constant factor) for the true δ-tolerancing problem.

4. An on-line decision of the containment problem

In the previous section we provided solutions to several variants of the offset-polygon maximum-cover and containment problems, under various rigid transformations. In this section we present an alternate "on-line" approach to offset-polygon containment decision problem for the translation-only case. As before, we assume convex polygons and deal with simple polygons in a later section. The idea of this on-line approach is that instead of being given the entire set S at once, the points are read one at a time, and for each new point we decide whether there is a placement of the annulus region of P that contains all the points *seen so far*. There are several motivations for the on-line approach. One is that for the decision problem we need not necessarily process the entire point set; if after a certain number of points there is no longer a placement containing them all then we can halt immediately and answer 'No' (thus offering some savings in running time over unnecessarily processing all the points). This may be particularly useful for the tolerancing problem. A second advantage is the ability to process incoming points *as they arrive* while simultaneously reading subsequent points (a form of pipelining). This is an advantage in the cases of the proposed applications where the points are not stored in a file but are read one-at-a-time by an external device. A third possible advantage is that as more points are read we can slowly *refine* the space of possible placements of P. This can be helpful for both the robot localization and geometric tolerancing problems where we might direct the input device for further measurements. Finally, the on-line approach allows for the *pruning* of the data structures providing a more efficient approach for most practical applications.

4.1. Basic algorithm approach

We begin with the basic ideas of the on-line approach. We want to read input points one at a time. For each point q_i we construct and store a data structure (similar to that of the algorithm in Fig. 3) that maintains optimal placements of the annulus region around P in contact with q_i. We also update the data

structures for the existing points q_j for $j < i$. That is, for each $j < i$ we: (1) compute the translations that keep the annulus region of P in contact with q_i and contain q_j, and add this information to the new data structure of q_i; and (2) compute translations that keep the annulus region in contact with q_j and contain q_i, and update the data structure of q_j. Remember that for each point q_j these translations are computed from the intersections of the translated offset polygons in $O(\log m)$ time by Lemma 2. However, our use of data structures for the on-line algorithm differs in two ways from the original algorithm. The first difference is that (unfortunately) we need to store several data structures simultaneously, rather than computing the optimal placement for one point and then discarding it. This is because each data structure is continuously being updated as new points are added. The second difference is more advantageous: since we are concerned only with the decision problem of whether there is a placement containing *all* n points, we need keep track of only those placements containing all points seen so far. Any placement that does not contain all points can be discarded. That is, we want the *intersections* of all the pairwise containing regions, where each region is given by a pair of segments (possibly empty) on the inner offset polygon and another pair of segments (also possibly empty) on the outer offset polygon. If at any point in the algorithm there are no such remaining placements, then we can halt and output 'No'.

4.2. Analysis and details of data structure

How do we store the set of placements containing all points? Recall that the region of placements containing q_i and with q_j on the boundary corresponds to a pair of segments along the inner and outer boundaries of the annulus region. For each point, we store these placements in two balanced binary-search trees (one for the inner polygon and one for the outer polygon) ordered clockwise around the boundary of the polygon. Unfortunately, it is possible to construct a case where the complexity of the set of placements containing all points is $\Theta(n)$. (Each new pair of segments increases the complexity of the set by 2.) Thus the space required *per point* may be as high as $\Theta(n)$ for a total of $\Theta(n^2)$ space. The searches, inserts and deletes can all be performed in $O(\log n)$ time. In particular, for each new point q_i added to the structure of point q_j, there are at most two segments to be added to both the inner and outer offset polygons. Since we want only placements containing all points, we store the *intersections* of these two new segments with all existing segments. We find the segment endpoints in $O(\log m)$ time and delete all regions not inside the endpoints. Deleting one segment and rebalancing the tree requires $O(\log n)$ time. The total number of insertions and deletions to all trees (per some point q_i) is $O(n)$ with a total of $O(n^2)$ tree operations. The overall complexity is thus $O(n^2 \log(nm) + m)$ time and $O(n^2 + m)$ space in the worst case (when no pruning is done). The algorithm may terminate early with a 'No' answer.

4.3. Improvement by pruning

Both the space and time complexity of the algorithm can be improved considerably by on-line pruning. Recall from the previous section that for each point q_i we need to store only the placements containing *all* the points: that is, the intersections of all $i - 1$ intersection regions. We can discard the data structure of a point q_i when this intersection region becomes empty. This happens when there are no longer any placements containing all other points with q_i on the boundary.

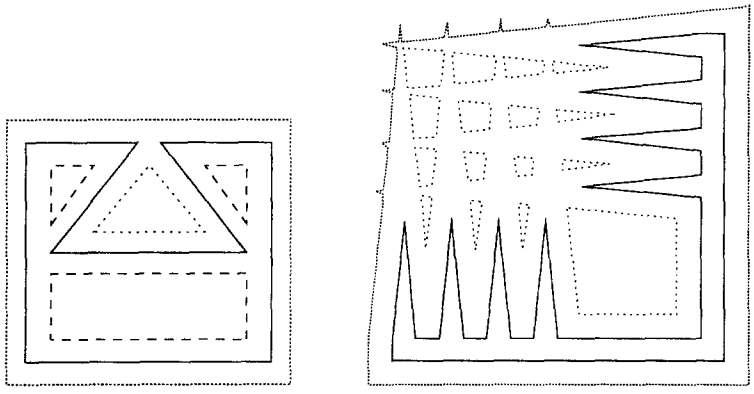
We define $H_1 = \{p_1\}$ and H_i (for $2 \leqslant i \leqslant n$) to contain all the points $x \in H_{i-1} \cup \{p_i\}$ such that there exists a placement $\tau(P)$ which contains $H_{i-1} \cup \{p_i\}$ with x on the boundary of $\tau(P)$. Let $h_i = |H_i|$. (In case $h_i = 0$ for some i the algorithm terminates with a 'No' answer.) Also, let h be the maximum value

of h_i for $1 \leqslant i \leqslant n$. Then the total number of data structures after the ith step of the algorithm is $O(h_i)$ and the total at any time is $O(h)$. The total time required to update existing data structures for a new point q_i is $O(h_{i-1} \log n) = O(h \log n)$.

One might now ask whether we can discard point q_i altogether. Unfortunately, it is possible for a point q_i which is not on the boundary of any placement containing the first i points to be the *only* point not contained in some later placement. This suggests that though we can remove the data structure of q_i, the point q_i still needs to be considered for updating the data structures of later points q_j (for $j > i$). We get around this problem in the following way. Whenever the data structure for point q_i is discarded, we also *mark* the point q_i for deletion. We create a queue that keeps discarded points in order of deletion, and also maintains for each point the step in the algorithm in which it was marked deleted. The algorithm then continues as before, but for each new point q_i, we only update its data structure by those points q_j (with $j < i$) whose own data structures are still active (not marked for deletion). Thus point q_i not only updates only $O(h_{i-1})$ data structures, but its data structure is updated by only $O(h_{i-1})$ points for a total of $O(h \log(nm))$ time per new point. With this change the algorithm proceeds in the same way, processing one point at a time and terminating if there are no possible placements containing all points. If no placement is possible, the algorithm can correctly terminate and answer 'No'. However if all n points are successfully processed, then there may be more work to do before a 'Yes' answer can be returned, because we have not yet fully considered points marked for deletion. We return to the queue of points marked for deletion. Suppose the first point on the queue, q_k, was discarded when point q_i was processed. If there exists a data structure for a point q_j (with $j < i$) then all points have been added to this data structure; that is, any placement represented by this data structure contains all points. We can halt and output 'Yes'. Otherwise, we undelete q_k and update all the remaining active data structures. This process is repeated until either there are no remaining data structures (then we halt and answer 'No') or until there is a data structure to which all points have been added (where we halt and answer 'Yes'). In either case, in this final step we add $O(n)$ points from the queue to $O(h)$ remaining structures in $O(nh \log(nm))$ time. The total running time for the algorithm is therefore $O(nh \log(nm) + m)$ time, and the required space is only $O(nh + m)$ in the worst case.

It is interesting to ask what the expected values for h are. Note that a point cannot be in contact with the outer offset polygon unless it is on the convex hull of the contained points. The conditions for when a point can appear in contact with the inner offset polygon are similar but slightly more complex. We conjecture, therefore, that similar to known results for convex hulls the expected value for h is $O(\log n)$ if the points are uniformly distributed within the annulus region or have a uniform distance from the polygon P. If the distance of points from P follows a normal distribution, then the expected value of h may be even smaller, such as $O(\log \log n)$. In the former case the on-line algorithm would require $O(n \log n \log(nm) + m)$ time. The space and time complexities are improved even further in practice because the region of placements for a given point is unlikely to have the worst case linear complexity.

The correctness of the algorithm is maintained in this pruning approach. If we halt and answer 'No', then there is some subset of points which cannot be simultaneously all contained in any optimal placement. If there is no placement containing a subset, then there is no placement containing the entire set and 'No' is the correct answer. Conversely, the algorithm does not answer 'Yes' until it finds a containing placement to which all points have been added.



(a) (b)

Fig. 4. Simple polygons with nonsimple offsets. (a) Nonsimple inner and outer offsets. (b) Intersecting offsets of narrow spikes.

5. Simple polygons

In this section we extend our results to the case of simple polygons. [2] Fig. 4(a) shows a simple polygon (with solid edges) whose outer offset is *perforated*: it contains a boundary (dotted densely) and a hole (dotted sparsely). The inner offset of the same polygon consists of three distinct polygons (with dashed edges). Fig. 4(b) shows another simple polygon whose outer offset is perforated. For algorithm efficiency, we would like to disallow polygons with narrow corridors in which the inner or outer δ-offsets become disconnected or non-simply-connected. The following definition formalizes this restriction.

Definition 3 (δ-wide polygons). A *δ-wide* polygon P is a simple polygon with the property that if $p, q \in \partial P$ with $\text{dist}(p, q) \leqslant 2\delta$ then there is a path connecting p and q along the boundary of P such that every point on the path is at most 2δ away from p or every point is at most 2δ away from q.

This restriction is a reasonable one for many of the proposed applications. Remember that vertices of the input polygon represent features in the model, and the value of δ represents the error tolerance. What we are assuming is that the minimum feature size is at least double the error tolerance, which is a common limitation of manufacturing processes.

We can solve simple-polygon variants of the offset-polygon maximum-cover problem for δ-wide simple polygons with a slightly modified version of the algorithm given in Fig. 3. Let us use $\overline{O}_{P,\delta}$ to denote the *true* (nonlinear) outer δ-offset of P. Similarly, we call the inner curve formed by straight segments and circular arcs at distance δ the *true* inner δ-offset of P and denote it by $\overline{I}_{P,\delta}$. Note that $\overline{I}_{P,\delta}$ and $\overline{O}_{P,\delta}$ are each of complexity $O(m)$ for a δ-wide simple polygon P of size m. Instead of having at most two intersections between $\tau(\overline{I}_{P,\delta})$ and $\overline{O}_{P,\delta}$, we can have $\Theta(m^2)$ pairwise intersections in the worst case requiring $\Theta(m^2)$ time to compute (even in a brute-force way). Each pair of points has two offset polygons, each of which has $O(m^2)$ intersections with the polygon being swept. So the size of the queue

[2] See [1] for a discussion of the *straight skeleton* of a simple polygon which is closely related to the notion of the inner offset polygon. This discussion is not, however, directly related to the problems discussed in this paper.

is $O(nm^2)$ and queue operations can be performed in $O(\log(nm))$ time. The overall complexity of the algorithm becomes $O(n^2m^2\log(nm))$ time and $O(nm^2)$ space.

The above running times also hold for the linearized versions of the offset-polygon maximum-cover problem if we disallow polygons with narrow features that cause the outer or inner offset polygon to intersect itself. Note that for a general simple polygon P, both linearized outer and inner boundaries, $O_{P,\delta}$ and $I_{P,\delta}$, can contain some points further than δ from ∂P. As a result, $I_{P,\delta}$ and $O_{P,\delta}$ can each be of complexity $\Theta(m^2)$ for a polygon P with m vertices. Fig. 4(b) gives an illustration of this. Thus there can be $\Theta(m^4)$ intersections between $\tau(I_{P,\delta})$ and $O_{P,\delta}$. The simple polygons to which our algorithm applies must therefore be δ-wide without narrow spikes.

The on-line algorithm can also be modified for simple polygons. If we make the assumption that the features of the polygon are such that the annulus region has $O(m)$ complexity, then in the worst case the number of intersections between two translated copies of the annulus is $O(m^2)$ and the complexity of the arrangement of containing regions for a given point is $O(nm^2)$. The on-line algorithm therefore requires $O(n^2m^2\log(nm))$ time and $O(n^2m^2)$ space.

6. Conclusion

In this paper we provide efficient algorithms for polygon offset placement problems. We handle the convex and nonconvex cases, the translation-only variant as well as the translation and rotation variant, the static and dynamic modes of input points, and decision and optimization versions of the problems. There are several possible further research directions which include the following.

1. Minimizing the value of δ such that the placement of the given polygon annulus contains some given value $k < n$ of them (offset-polygon partial containment).
2. Generalizing from a polygon to a collection of polygonal *chains*. (This variant often occurs in applications to robot localization.)
3. Generalizing from polygons to smooth shapes.
4. Computing *approximate* solutions to all of these problems.
5. Proving lower bounds for the problems.
6. Solving similar problems in higher dimensions.
7. Analyzing the *expected* value of h in the pruning version of the on-line algorithm (Section 4.3).

Appendix A. Proof of Theorem 1

Theorem 1. *Given a polygon P, a distance δ, and a translation τ, the (linearized) offset polygons $\tau(I_{P,\delta})$ and $O_{P,\delta}$ intersect at most twice, where each intersection may be a point or (in the degenerate case) a segment.*

Proof. Assume without loss of generality that τ is parallel to the positive direction of the x-axis. Let ℓ_1 and ℓ_2 be the upper and lower parallel supporting-lines of $I_{P,\delta}$ in the direction of τ (see Fig. 5(a)). Let t and b be the top and bottom vertices of $I_{P,\delta}$, respectively (the points of tangency with ℓ_1 and ℓ_2). Let t' and b' be the points of intersection between ℓ_1 and ℓ_2 and the right side of $O_{P,\delta}$. Finally, let I_l and I_r be

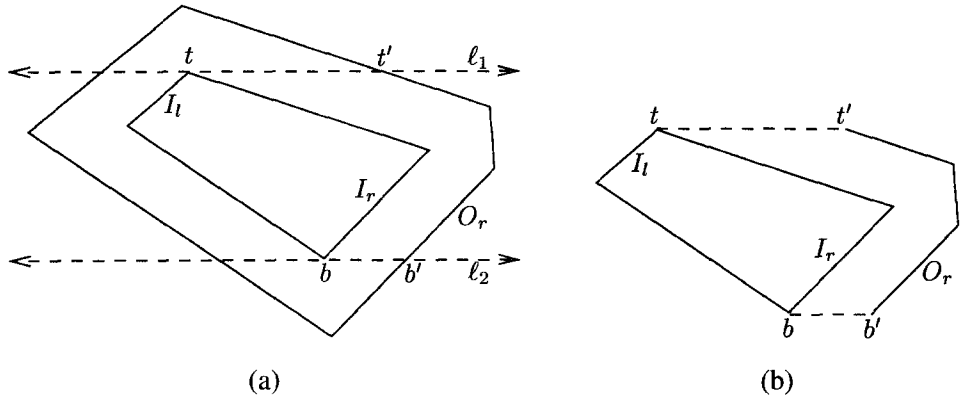


Fig. 5. Critical chains that might intersect.

the left and right chains of $I_{P,\delta}$ determined by the points b and t, and let O_r be the right chain of $O_{P,\delta}$ (connecting t' and b').

We observe that the translation of $I_{P,\delta}$ remains in the horizontal slab between ℓ_1 and ℓ_2. Thus the only place where it can intersect $O_{P,\delta}$ is on O_r. We can thus simplify our diagram to Fig. 5(b). We need now only show that $\tau(I_r)$ can intersect O_r at most twice, and that $\tau(I_l)$ can intersect O_r at most twice, and furthermore that if $\tau(I_r)$ intersects O_r more than once then $\tau(I_l)$ does not intersect O_r at all and vice versa: if $\tau(I_l)$ intersects O_r more than once then $\tau(I_r)$ does not intersect O_r at all. We can derive all these claims from the following observations.

First, consider any intersection point x' between $\tau(I_r)$ and O_r. Since x' is on $\tau(I_r)$, it must be the translation $\tau(x)$ of a point x on I_r. But x' also lies on O_r. Thus we conclude that intersections between $\tau(I_r)$ and O_r occur precisely at points where O_r is exactly a (horizontal) distance of τ from I_r. How many such points are there on O_r? Consider the function that, for values of y ranging from ℓ_1 down to ℓ_2, gives the distance in the τ direction from I_r to O_r. Because these are inner and outer offset polygons of the same polygon P, this is a weakly monotone decreasing function on y as y decreases from ℓ_1 down to the rightmost point of I_r. Similarly, as y continues from this rightmost point of I_r down to ℓ_2, it is a weakly monotone increasing function. But in a monotone function, a value can only appear once (or in one continuous range). Since the function is divided into two weakly monotone regions, a fixed value can appear at most twice (or in two continuous ranges). This suffices to show that there can be at most two intersections between $\tau(I_r)$ and O_r, one in each weakly monotone region of the distance function. Note that each intersection can be either a single point or a connected portion of an edge. The edge intersection possibility follows from the fact that the functions are weakly monotone rather than strictly monotone.

Next we note that the same holds for I_l and O_r except that with the left chain of $I_{P,\delta}$, the distance function is first weakly monotone increasing, and then weakly monotone decreasing, as y goes down from ℓ_1 to ℓ_2. Thus there are at most two intersections (points or connected portions of segments) between I_r and O_r, and at most two intersections between I_l and O_r. It remains only to show that if $\tau(I_r)$ intersects O_r twice, then $\tau(I_l)$ does not intersect O_r at all, and vice versa. For this we need note only that in order for $\tau(I_r)$ to intersect O_r twice, τ must be less than the minimum of the distances from b to b' and from t to t', which are the largest values of the two monotone portions of the width function. But this condition implies that $\tau(I_l)$ does not intersect O_r at all. Similarly, for I_l to intersect O_r twice

we need τ to be greater than the maximum of the distances from b to b' and from t to t', which would preclude an intersection between $\tau(I_r)$ and O_r. □

References

[1] O. Aichholzer, D. Alberts, F. Aurenhammer, B. Gärtner, A novel type of skeleton for polygons, J. of Universal Computer Science (an electronic journal) 1 (1995) 752–761.
[2] P.K. Agarwal, M. Sharir, Efficient randomized algorithms for some geometric optimization problems, Discrete Comput. Geom. 16 (1996) 317–337.
[3] P.K. Agarwal, M. Sharir, S. Toledo, Applications of parametric searching in geometric optimization, J. Algorithms 17 (1994) 292–318.
[4] G. Barequet, M. Dickerson, M.T. Goodrich, Voronoi diagrams for medial-axis distance functions, in: Proc. 5th Workshop on Algorithms and Data Structures, Halifax, Nova Scotia, Canada, 1997, pp. 200–209.
[5] G. Barequet, M. Dickerson, P. Pau, Translating a convex polygon to contain a maximum number of points, Computational Geometry 8 (1997) 167–179.
[6] E.-C. Chang, C.-K. Yap, Issues in the metrology of geometric tolerancing, manuscript, Courant Institute of Mathematical Sciences, New York University.
[7] B. Chazelle, The polygon placement problem, in: F. Preparata (Ed.), Advances in Computing Research, Vol. 1, JAI Press, 1983, pp. 1–34.
[8] I.J. Cox, J.B. Kruskal, Determining the 2- or 3-dimensional similarity transformation between a point set and a model made of lines and arcs, in: Proc. 28th Conf. on Decision and Control, 1989, pp. 1167–1172.
[9] C.A. Duncan, M.T. Goodrich, E.A. Ramos, Efficient approximation and optimization algorithms for computational metrology, in: Proc. 8th Ann. ACM-SIAM Symp. on Discrete Algorithms, New Orleans, LA, 1997, pp. 121–130.
[10] M. Dickerson, D. Scharstein, Optimal placement of convex polygons to maximize point containment, in: Proc. 7th Ann. ACM-SIAM Symp. on Discrete Algorithms, Atlanta, GA, 1996, pp. 114–121.
[11] W.P. Dong, E. Mainsah, P.F. Sullivan, K.F. Stout, Instruments and measurement techniques of 3-dimensional surface topography, in: K.F. Stout (Ed.), Three-Dimensional Surface Topography: Measurement, Interpretation and Applications, Penton Press, Bristol, PA, 1994.
[12] A. Efrat, M. Sharir, A. Ziv, Computing the smallest k-enclosing circle and related problems, Computational Geometry 4 (1994) 119–136.
[13] D. Eppstein, J. Erickson, Iterated nearest neighbors and finding minimal polytopes, Discrete Comput. Geom. 11 (1994) 321–350.
[14] L. Guibas, R. Motwani, P. Raghavan, The robot localization problem, in: Algorithmic Foundations of Robotics, A.K. Peters, 1995, pp. 269–282.
[15] M.E. Houle, G.T. Toussaint, Computing the width of a set, in: Proc. 1st Ann. ACM Symp. on Computational Geometry, 1985, pp. 1–7.
[16] D.P. Huttenlocher, S. Ullman, Recognizing solid objects by alignment with an image, Internat. J. Comput. Vision 5 (1990) 195–212.
[17] D. Kirkpatrick, J. Snoeyink, Tentative prune-and-search for computing fixed-points with applications to geometric computation, Fund. Inform. 22 (1995) 353–370.
[18] V.B. Le, D.T. Lee, Out-of-roundness problem revisited, IEEE Trans. Pattern Anal. Machine Intell. 13 (1991) 217–223.
[19] H.P. Lenhof, M. Smid, Sequential and parallel algorithms for the k closest pairs problem, Internat. J. Comput. Geom. Appl. 5 (1995) 273–288.
[20] A.A.G. Requicha, Mathematical meaning and computational representation of tolerance specifications, in: Proc. Internat. Forum on Dimensional Tolerancing and Metrology, 1993, pp. 61–68.

[21] M. Smid, R. Janardan, On the width and roundness of a set of points in the plane, in: Proc. 7th Canadian Conf. on Computational Geometry, Québec City, Québec, Canada, 1995, pp. 193–198.
[22] V. Srinivasan, Role of sweeps in tolerance semantics, in: Proc. Internat. Forum on Dimensional Tolerancing and Metrology, 1993, pp. 69–78.
[23] K. Swanson, D.T. Lee, V.L. Wu, An optimal algorithm for roundness determination on convex polygons, Computational Geometry 5 (1995) 225–235.
[24] C.-K. Yap, Exact computational geometry and tolerancing metrology, in: D. Avis, P. Bose (Eds.), Snapshots of Computational and Discrete Geometry, Vol. 3, McGill School of Computer Science, 1995.